ЛЕКЦИЯ № 11. ЭКСПЛУАТАЦИЯ БАЗ ДАННЫХ. ЗАЩИТА БАЗ ДАННЫХ. УПРАВЛЕНИЕ ТРАНЗАКЦИЯМИ. УПРАВЛЕНИЕ ПАРАЛЛЕЛЬНЫМ ДОСТУПОМ. ЗАКЛЮЧЕНИЕ.

10.1 Понятие восстановления системы

Восстановление в системе управления базами данных, означает в первую очередь восстановление самой базы данных, т.е. возвращение базы данных в правильное состояние, если какой-либо сбой сделал текущее состояние неправильным или подозрительным. Основной принцип, на котором строится такое восстановление, – это избыточность. Избыточность организуется на физическом уровне. Такая избыточность будет скрыта от пользователя, а следовательно, не видна на логическом уровне. Другими словами, если любая часть информации, содержащаяся в базе данных, может быть реконструирована из другой хранимой в системе избыточной информации, значит, база данных восстанавливаема.

10.2 Транзакции

10.2.1 Понятие транзакции

Транзакция – это логическая единица работы. Например. Предположим сначала, что отношение Students (отношение студентов) включает дополнительный атрибут AvgMark, представляющий собой средний балл студента, по результатам сдачи текущей сессии. Значение AvgMark для любой определенной детали предполагается равным среднему арифметическому всех значений Mark из таблицы Marks для всех оценок полученных в текущем семестре.

В приведенном примере предполагается, что речь идет об одиночной, атомарной операции. На самом деле добавление новой оценки в таблицу Marks – это выполнение двух обновлений в базе данных (под обновлениями здесь, конечно, понимаются операции insert, delete, а также сами по себе операции update). Более того, в базе данных между этими двумя обновлениями временно нарушается требование, что значение AvgMark для студента 1 равно среднему арифметическому всех значений поля Mark для студента 1 в текущем семестре. Таким образом, логическая единица работы (т.е. транзакция) – не просто одиночная операция системы баз данных, а скорее согласование нескольких таких операций. В общем, это преобразование одного согласованного состояния базы данных в другое, причем в промежуточных точках база данных находится в несогласованном состоянии.

Из этого следует, что недопустимо, чтобы одно из обновлений было выполнено, а другое нет, так как база данных останется в несогласованном состоянии. В идеальном случае должны быть выполнены оба обновления. Однако нельзя обеспечить стопроцентную гарантию, что так и будет. Не исключена вероятность того, что, система,

например, будет разрушена между двумя обновлениями, или же на втором обновлении произойдет арифметическое переполнение и т.п. Система, поддерживающая транзакции, гарантирует, что если во время выполнения неких обновлений произошла ошибка (по любой причине), то все эти обновления будут аннулированы. Таким образом, транзакция или выполняется полностью, или полностью отменяется (как будто она вообще не выполнялась).

Системный компонент, обеспечивающий атомарность (или ее подобие), называется администратором транзакций (или диспетчером транзакций), а ключами к его выполнению служат операторы COMMIT TRANSACTION и ROLLBACK TRANSACTION.

Оператор COMMIT TRANSACTION (для краткости commit) сигнализирует об успешном окончании транзакции. Он сообщает администратору транзакций, что логическая единица работы завершена успешно, база данных вновь находится (или будет находиться) в согласованном состоянии, а все обновления, выполненные логической единицей работы, теперь могут быть зафиксированы, т.е. стать постоянными.

Оператор ROLLBACK TRANSACTION (для краткости ROLLBACK) сигнализирует о неудачном окончании транзакции. Он сообщает администратору транзакций, что произошла какая-то ошибка, база данных находится в несогласованном состоянии и все обновления могут быть отменены, т.е. аннулированы.

Для отмены обновлений система поддерживает файл регистрации, или журнал, на диске, где записываются детали всех операций обновления, в частности новое и старое значения модифицированного объекта. Таким образом, при необходимости отмены некоторого обновления система может использовать соответствующий файл регистрации для возвращения объекта в первоначальное состояние.

Еще один важный момент. Система должна гарантировать, что индивидуальные операторы сами по себе атомарные (т.е. выполняются полностью или не выполняются совсем). Это особенно важно для реляционных систем, в которых операторы многоуровневые и обычно оперируют множеством кортежей одновременно; такой оператор просто не может быть нарушен посреди операции и привести систему в несогласованное состояние. Другими словами, если произошла ошибка во время работы такого оператора, база данных должна остаться полностью неизмененной. Более того, это должно быть справедливо даже в том случае, когда действия оператора являются причиной дополнительной, например каскадной, операции.

10.2.2 Восстановление транзакции.

Транзакция начинается с успешного выполнения оператора BEGIN TRANSACTION) и заканчивается успешным выполнением либо оператора COMMIT, либо ROLLBACK. Оператор COMMIT устанавливает так называемую точку фиксации (которая в коммерческих продуктах также называется точкой синхронизации (syncpoint). Точка фиксации соответствует концу логической единицы работы и, следовательно, точке, в которой база данных находится (или будет находиться) в состоянии согласованности. В противовес этому, выполнение оператора ROLLBACK вновь возвращает базу данных в состояние, в котором она была во время операции BEGIN TRANSACTION, т.е. в предыдущую точку фиксации.

Случаи установки точки фиксации:

1. Все обновления, совершенные программой с тех пор, как установлена предыдущая точка фиксации, выполнены, т.е. стали постоянными. Во время выполнения все такие обновления могут расцениваться только как пробные (в том смысле, что они могут быть не выполнены, например прокручены назад). Гарантируется, что однажды зафиксированное обновление так и останется зафиксированным (это и есть определение понятия "зафиксировано").

2. Все позиционирование базы данных утеряно, и все блокировки кортежей реализованы. Позиционирование базы данных здесь означает, что в любое конкретное время программа обычно адресована определенным кортежам. Эта адресуемость в точке фиксации теряется.

Следовательно, система может выполнить откат транзакции как явно – например по команде ПО с которым работает пользователь, так и неявно – для любой программы, которая по какой-либо причине не достигла запланированного завершения операций, входящих в транзакцию.

Из этого видно, что транзакции — это не только логические единицы работы, но также и единицы восстановления при неудачном выполнении операций. При успешном завершении транзакции система гарантирует, что обновления постоянно установлены в базе данных, даже если система потерпит крах в следующий момент. Возможно, что в системе произойдет сбой после успешного выполнения COMMIT, но перед тем, как, обновления будут физически записаны в базу данных (они все еще могут оставаться в буфере оперативной памяти и таким образом могут быть утеряны в момент сбоя системы). Даже если подобное случилось, процедура перезагрузки системы все равно должна устанавливать эти обновления в базу данных, исследуя соответствующие записи в файле регистрации. Из этого следует, что файл регистрации должен быть физически записан перед завершением операции COMMIT. Это важное правило ведения файла регистрации известно как протокол предварительной записи в журнал (т.е. запись об операции осуществляется перед ее выполнением). Таким образом, процедура перезагрузки сможет восстановить любые успешно завершенные транзакции, хотя их обновления не были записаны физически до аварийного отказа системы. Следовательно, как указывалось ранее, транзакция действительно является единицей восстановления.

10.2.3 Свойства АСИД.

Из предыдущих разделов следует, что транзакции обладают четырьмя важными свойствами: атомарность, согласованность, изоляция и долговечность (назовем это свойствами АСИД).

1. Атомарность. Транзакции атомарны (выполняется все или ничего).
2. Согласованность. Транзакции защищают базу данных согласованно. Это означает, что транзакции переводят одно согласованное состояние базы данных в другое без обязательной поддержки согласованности во всех промежуточных точках.
3. Изоляция. Транзакции отделены одна от другой. Это означает, что, если даже будет запущено множество конкурирующих друг с другом транзакций, любое обновление определенной транзакции будет скрыто от остальных до тех пор, пока эта транзакция выполняется. Другими словами, для любых двух отдаленных транзакций T1 и T2 справедливо следующее утверждение: T1 сможет увидеть обновление T2 только после выполнения T2, а T2 сможет увидеть обновление T1 только после выполнения T1.
4. Долговечность. Когда транзакция выполнена, ее обновления сохраняются, даже если в следующий момент произойдет сбой системы.

10.2.4 Алгоритм восстановления после сбоя системы

Система должна быть готова к восстановлению не только после небольших локальных нарушений, таких как невыполнение операции в пределах определенной транзакции, но также и после глобальных нарушений типа сбоев в питании вычислительного устройства и др. Местное нарушение по определению поражает только транзакцию, в которой оно собственно и произошло. Глобальное нарушение поражает сразу все транзакции и, следовательно, приводит к значительным для системы последствиям.

Существует два вида глобальных нарушений:

1. Отказы системы (например, сбои в питании), поражающие все выполняющиеся в данный момент транзакции, но физически не нарушающие базу данных в целом. Такие нарушения в системе также называют аварийным отказом программного обеспечения.

2. Отказы носителей (например, поломка головок дискового накопителя), которые могут представлять угрозу для базы данных или для какой-либо ее части и поражать. по крайней мере, те транзакции, которые используют эту часть базы данных. Отказы носителей также называют аварийным отказом аппаратуры.

10.2.5 Восстановление после отказов системы

Критической точкой в отказе системы является потеря содержимого оперативной памяти (в частности, рабочих буферов базы данных). Поскольку точное состояние какой-либо выполняющейся в момент нарушения транзакции не известно, транзакция может не завершиться успешно и, таким образом, будет отменена при перезагрузке системы.

Более того, возможно, потребуется повторно выполнить определенную успешно завершившуюся до аварийного отказа транзакцию при перезагрузке системы, если не были физически выполнены обновления этой транзакции.

Для определения во время перезагрузки, какую транзакцию отменить, а какую выполнить повторно система в некотором предписанном интервале (когда в журнале накапливается определенное число записей) автоматически принимает контрольную точку. Принятие контрольной точки включает физическую запись содержимого рабочих буферов базы данных непосредственно в базу данных и специальную физическую запись контрольной точки, которая предоставляет список всех осуществляемых в данный момент транзакций. На рис. 0.1 рассматривается пять возможных вариантов выполнения транзакций до аварийного сбоя системы.

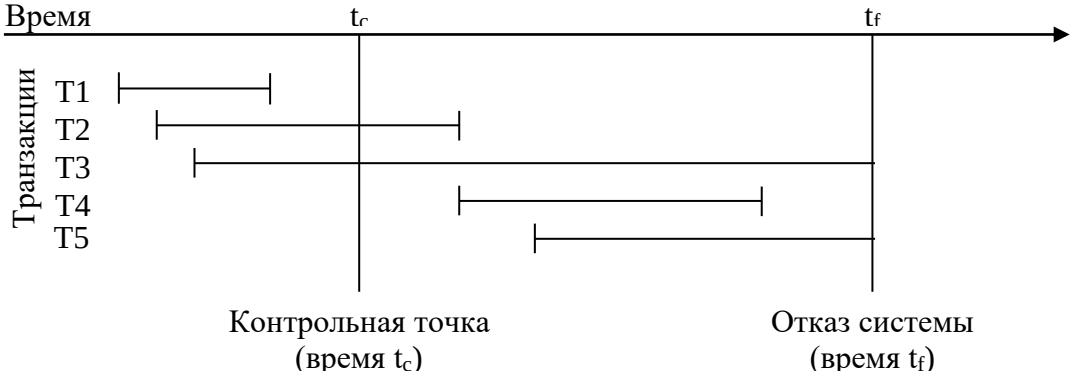

рис. 0.1 Варианты выполнения пяти транзакций.

Пояснения к рис. 0.1:

1. Отказ системы произошел в момент времени tf.

2. Близлежащая к моменту времени tf контрольная точка была принята в момент времени tc.

3. Транзакция T1 успешно завершена до момента времени tc.

4. Транзакция T2 начата до момента времени tc и успешно завершена после момента времени tc, но до момента времени tf.

5. Транзакция T3 также начата до момента времени tc, но не завершена к моменту времени tf

6. Транзакция T4 начата после момента времени tc и успешно завершена до момента времени tf.

7. Транзакция T5 также начата после момента времени tc, но не завершена к моменту времени tf.

Очевидно, что при перезагрузке системы транзакции типа Т3 и Т5 должны быть отменены, а транзакции типа Т2 и Т4 – выполнены повторно. Тем не менее заметьте, что транзакции типа Т1 вообще не включаются в процесс перезагрузки, так как обновления попали в базу данных еще до момента времени tc (т.е. зафиксированы еще до принятия контрольной точки). Отметьте также, что транзакции, завершившиеся неудачно (в том числе отмененные) перед моментом времени tf, вообще не будут вовлечены в процесс перезагрузки.

10.3 Параллелизм. Проблемы параллелизма

Термин параллелизм означает возможность одновременной обработки в СУБД многих транзакций с доступом к одним и тем же данным, причем в одно и то же время. В такой системе для корректной обработки параллельных транзакций без возникновения конфликтных ситуаций необходимо использовать некоторый метод управления параллелизмом.

Каждый метод управления параллелизмом предназначен для решения некоторой конкретной задачи. Тем не менее, при обработке правильно составленных транзакций возникают ситуации, которые могут привести к получению неправильного результата из-за взаимных помех среди некоторых транзакций. (Обратите внимание, что вносящая помеху транзакция сама по себе может быть правильной. Неправильный конечный результат возникает по причине бесконтрольного чередования операций из двух правильных транзакций). Основные проблемы, возникающие при параллельной обработке транзакций следующие:

1. проблема потери результатов обновления;
2. проблема незафиксированной зависимости;
3. проблема несовместимого анализа.

10.3.1 Проблема потери результатов обновления

Рассмотрим ситуацию, показанную на

рис. 0.**2**, в такой интерпретации: транзакция А извлекает некоторый кортеж р в момент времени t1; транзакция В извлекает некоторый кортеж р в момент времени t2; транзакция А обновляет некоторый кортеж р (на основе значений, полученных в момент времени t1) в момент времени t3; транзакция В обновляет тот же кортеж р (на основе значений, полученных в момент времени t2, которые имеют те же значения, что и в момент времени t1) в момент времени t4. Однако результат операции обновления, выполненной транзакцией А, будет утерян, поскольку в момент времени t4 она не будет учтена и потому будет "отменена" операцией обновления, выполненной транзакцией В.

Транзакция А	Время	Транзакция В
Извлечение кортежа р	t1	–
–	t2	Извлечение кортежа р
Обновление кортежа р	t3	–
–	t4	Обновление кортежа р

рис. 0.2. Потеря в момент времени t4 результатов обновления, выполненного транзакцией А.

10.3.2 Проблема незафиксированной зависимости

Проблема незафиксированной зависимости появляется, если с помощью некоторой транзакции осуществляется извлечение (или, что еще хуже, обновление) некоторого кортежа, который в данный момент обновляется другой транзакцией, но это обновление еще не закончено. Таким образом, если обновление не завершено, существует некоторая

вероятность того, что оно не будет завершено никогда. (Более того, в подобном случае может быть выполнен возврат к предыдущему состоянию кортежа с отменой выполнения транзакции.) В таком случае, в первой транзакции будут принимать участие данные, которых больше не существует. Эта ситуация показана на

рис. 0.**3**,

рис. 0.**4**.

В первом примере (

рис. 0.**3**) транзакция A в момент времени t2 встречается с невыполненным обновлением (оно также называется невыполненным изменением). Затем это обновление отменяется в момент времени t3. Таким образом, транзакция A выполняется на основе фальшивого предположения, что кортеж p имеет некоторое значение в момент времени t2, тогда как на самом деле он имеет некоторое значение, существовавшее еще в момент времени t1. В итоге после выполнения транзакции A будет получен неверный результат. Кроме того, обратите внимание, что отмена выполнения транзакции B может произойти не по вине транзакции B, а, например, в результате краха системы. (К этому времени выполнение транзакции A может быть уже завершено, а потому крушение системы не приведет к отмене выполнения транзакции A.)

Транзакция A	Время	Транзакция B
–	t1	Обновление кортежа p
Извлечение кортежа p	t2	–
–	t3	Отмена выполнения транзакции

рис. 0.3. Транзакция A становится зависимой от невыполненного изменения в момент времени t2.

Транзакция A	Время	Транзакция B
–	t1	Обновление кортежа p
Обновление кортежа p	t2	–
–	t3	Отмена выполнения транзакции

рис. 0.4. Транзакция A обновляет невыполненное изменение в момент времени t2, и результаты этого обновления утрачиваются в момент времени t3.

Второй пример, приведенный на

рис. 0.**4**, иллюстрирует другой случай. Не только транзакция A становится зависимой от изменения, не выполненного в момент времени t2, но также в момент времени t3 фактически утрачивается результат обновления, поскольку отмена выполнения транзакции B в момент времени t3 приводит к восстановлению кортежа p к исходному значению в момент времени t1. Это еще один вариант проблемы потери результатов обновления.

10.3.3 Проблема несовместимого анализа

На

рис. 0.**5** показаны транзакции A и B, которые выполняются для кортежей со счетами (**Ошибка! Источник ссылки не найден.**). При этом транзакция A суммирует балансы, транзакция B производит перевод суммы 10 со счета 3 на счет 1. Полученный в итоге транзакции A результат 110, очевидно, неверен, и если он будет записан в базе данных, то в ней может возникнуть проблема несовместимости. В таком случае говорят, что транзакция A встретилась с несовместимым состоянием и на его основе был выполнен

несовместимый анализ. Обратите внимание на следующее различие между этим примером и предыдущим: здесь не идет речь о зависимости транзакции A от транзакции B, так как транзакция B выполнила все обновления до того, как транзакция A извлекла СЧЕТ 3.

табл. 0.1 Остатки на счетах до выполнения транзакций.

Счет	СЧЕТ 1	СЧЕТ 2	СЧЕТ 3
Остаток	40	50	30

Транзакция A	Время	Транзакция B
Извлечение кортежа СЧЕТ 1: СУММА = 40	t1	–
Извлечение кортежа СЧЕТ 1: СУММА = 90	t2	–
–	t3	Извлечение кортежа СЧЕТ 3:
–	t4	Обновление кортежа СЧЕТ 3: $30 \rightarrow 20$
–	t5	Извлечение кортежа СЧЕТ 1:
–	t6	Обновление кортежа СЧЕТ 1: $40 \rightarrow 50$
–	t7	Завершение выполнения транзакции
Извлечение кортежа СЧЕТ 3: СУММА = 110 (а не 120)	t8	–

рис. 0.5. Транзакция A выполнила несовместимый анализ.

10.4 Понятие блокировки

Описанные выше проблемы могут быть разрешены с помощью методики управления параллельным выполнением процессов под названием блокировка. Ее основная идея очень проста: в случае, когда для выполнения некоторой транзакции необходимо, чтобы некоторый объект (обычно это кортеж базы данных) не изменялся непредсказуемо и без ведома этой транзакции (как это обычно бывает), такой объект блокируется. Таким образом, эффект блокировки состоит в том, чтобы "заблокировать доступ к этому объекту со стороны других транзакций", а значит, предотвратить непредсказуемое изменение этого объекта. Следовательно, первая транзакция в состоянии выполнить всю необходимую обработку с учетом того, что обрабатываемый объект остается в стабильном состоянии настолько долго, насколько это нужно.

Предположим, что в системе поддерживается два типа блокировок: блокировка без взаимного доступа (монопольная блокировка), называемая X-блокировкой (X locks – exclusive locks), и блокировка с взаимным доступом, называемая S-блокировкой (S locks - Shared locks). Замечание. X- и S-блокировки иногда называют блокировками записи и чтения соответственно. Предположим, что X- и S-блокировки единственно возможные, хотя в коммерческих системах существуют блокировки других типов. Кроме того, допустим, что в кортежи являются единственным типом "блокируемого объекта", хотя опять же н в коммерческих системах могут блокироваться и другие объекты. Ниже показано функционирование механизма блокировок.

1. Если транзакция A блокирует кортеж p без возможности взаимного доступа (X-блокировка), то запрос другой транзакции B с блокировкой этого кортежа p будет отменен.

2. Если транзакция A блокирует кортеж p с возможностью взаимного доступа (S-блокировка), то:

2.1. запрос со стороны некоторой транзакции B на X-блокировку кортежа будет отвергнут;

2.2. запрос со стороны некоторой транзакции B на S-блокировку кортежа p будет принят (т.е. транзакция B также будет блокировать кортеж p с помощью S-блокировки).

Эти правила можно наглядно представить в виде матрицы совместимости, показанной на

рис. 0.**6**, и интерпретировать ее следующим образом. Рассмотрим некоторый кортеж p и предположим, что транзакция A блокирую кортеж p различными типами блокировки (это обозначено соответствующими символами S и X, а отсутствие блокировки — прочерком). Предположим также, что некоторая транзакция B запрашивает блокировку кортежа p, что обозначено в первом слева столбце матрицы на

рис. 0.**6** (для полноты картины в таблице также приведен случай "отсутствия блокировки"). В других ячейках матрицы символ N обозначает конфликтную ситуацию (запрос со стороны транзакции B не может быть удовлетворен, и сама эта транзакция переходит в состояние ожидания), а Y – полную совместимость (запрос со стороны транзакции B удовлетворен). Очевидно, что эта матрица является симметричной.

	X	S	–
X	N	N	Y
S	N	Y	Y
–	Y	Y	Y

рис. 0.6. Матрица совместимости для X- и S-блокировки.

Введем протокол доступа к данным, который на основе введения только что описанных X- и S-блокировки позволяет избежать возникновения проблем параллелизма.

1. Транзакция, предназначенная для извлечения кортежа, прежде всего должна наложить S-блокировку на этот кортеж.

2. Транзакция, предназначенная для обновления кортежа, прежде всего должна наложить X–блокировку на этот кортеж. Иначе говоря, если, например, для последовательности действий типа извлечение/обновление для кортежа уже задана S-блокировка, то ее необходимо заменить X-блокировкой. Блокировки в транзакциях обычно задаются неявным образом: например, запрос на "извлечение кортежа" является неявным запросом с S-блокировкой, а запрос на "обновление кортежа" – неявным запросом с X-блокировкой соответствующего кортежа. При этом под термином "обновление" (как и ранее) подразумеваются помимо самих операций обновления также операции вставки и удаления.

3. Если запрашиваемая блокировка со стороны транзакции B отвергается из-за конфликта с некоторой другой блокировкой со стороны транзакции A, то транзакция B переходит в состояние ожидания. Причем транзакция B будет находиться в состоянии ожидания до тех пор, пока не будет снята блокировка, заданная транзакцией A. В системе обязательно должны быть предусмотрены способы устранения бесконечно долгого состояния ожидания транзакции B.

4. X–блокировки сохраняются вплоть до конца выполнения транзакции (до операции "завершение выполнения" или "отмена выполнения"). S-блокировки также обычно сохраняются вплоть до этого момента.

10.5 Решение проблем параллелизма

Рассмотрим решение проблем параллелизма с помощью механизма блокировок.

10.5.1 Проблема потери результатов обновления.

На

рис. 0.**7** приведена измененная версия процесса, показанного на

рис. 0.**2**, с учетом применения протокола блокировки для чередующихся операций. Операция обновления для транзакции A в момент времени t3 не будет выполнена, поскольку она является неявным запросом с заданием X-блокировки для кортежа p, а этот запрос вступает в конфликт с S-блокировкой, уже заданной транзакцией B. Таким образом, транзакция A переходит в состояние ожидания. По аналогичным причинам транзакция B переходит в состояние ожидания в момент времени t4.Обновления теперь не утрачиваются, однако возникает новая проблема – бесконечное ожидание или тупиковая ситуация. Способы решения этой проблемы рассматриваются ниже.

Транзакция A	Время	Транзакция B
Извлечение кортежа p (задание S-блокировки для p)	t1	–
–	t2	Извлечение кортежа p (задание S-блокировки для p)
Обновление кортежа p (задание X-блокировки для p)	t3	–
Ожидание	t4	Обновление кортежа p (задание X-блокировки для p)
Ожидание		Ожидание

рис. 0.7. Хотя обновления не утрачиваются, но в момент времени t4 возникает тупиковая ситуация.

10.5.2 Проблема незафиксированной зависимости.

На

рис. 0.**8**,

рис. 0.**9** приведены в измененном виде примеры, показанные ранее на

рис. 0.**3** и

рис. 0.**4** соответственно. Они демонстрируют чередующееся выполнение операций согласно описанному выше протоколу блокировки. Операция для транзакции A в момент времени t2 (извлечение на

рис. 0.**8** и обновление на

рис. 0.**9**) не будет выполнена. Дело в том, что она является неявным запросом с заданием блокировки для кортежа p, а этот запрос вступает в конфликт с X-блокировкой, уже заданной транзакцией B. Таким образом, транзакция A переходит в состояние ожидания до тех пор, пока не будет прекращено выполнение транзакции B (до операции окончания или отмены выполнения транзакции B). Тогда заданная транзакцией B блокировка будет снята и транзакция A может быть выполнена. Причем транзакция A будет иметь дело с некоторым фиксированным значением (либо существовавшим до выполнения транзакции B при отмене ее выполнения, либо полученным после выполнения транзакции B). В любом случае транзакция A больше не зависит от незафиксированного обновления.

Транзакция A	Время	Транзакция B
–	t1	Обновление кортежа p (задание X-блокировки для p)
Извлечение кортежа p (задание S-блокировки для p)	t2	–
Ожидание	t3	Отмена выполнения транзакции (снятие X-блокировки для p)
Итог: Извлечение кортежа p (задание S-блокировки для p)	t4	

рис. 0.8. Транзакция A предохраняется от выполнения операций с незафиксированным изменением в момент времени t2.

Транзакция A	Время	Транзакция B
–	t1	Обновление кортежа p (задание X-блокировки для p)
Обновление кортежа p (задание X-блокировки для p)	t2	–
Ожидание	t3	Отмена выполнения транзакции (снятие X-блокировки для p)
Итог: Обновление кортежа p (задание X-блокировки для p)	t4	

рис. 0.9. Транзакция A предохраняется от выполнения операций с незафиксированным изменением в момент времени t2.

10.5.3 Проблема несовместимого анализа

На

рис. 0.**10** приведена измененная версия отношения (

рис. 0.**5**) с перечислением чередующихся транзакций согласно протоколу блокировки. Операция обновления для транзакции B в момент времени t6 не будет выполнена. Дело в том, что она является неявным запросом с заданием X-блокировки для кортежа СЧЕТ 1, а этот запрос вступает в конфликт с S-блокировкой, уже заданной транзакцией A. Таким образом, транзакция B переходит в состояние ожидания. Точно так же операция извлечения для транзакции A в момент времени t7 не будет выполнена. Дело в том, что она является неявным запросом с заданием S-блокировки для кортежа СЧЕТ 3, а этот запрос вступает в конфликт с X-блокировкой, уже заданной транзакцией B. Таким образом, транзакция A переходит в состояние ожидания. Следовательно, блокировка хотя и помогает решить одну проблему (а именно проблему несовместимого анализа), но приводит к необходимости решения другой проблемы (а именно проблемы возникновения тупиковой ситуации).

СЧЕТ 1 40	СЧЕТ 2 50	СЧЕТ 3 30
Транзакция А	Время	Транзакция В
Извлечение кортежа СЧЕТ 1: (задание S-блокировки для СЧЕТ 1) СУММА = 40	t1	–
Извлечение кортежа СЧЕТ 1: (задание S-блокировки для СЧЕТ 2) СУММА = 90	t2	–
–	t3	Извлечение кортежа СЧЕТ 3: (задание S-блокировки для СЧЕТ 3)
–	t4	Обновление кортежа СЧЕТ 3: (задание X-блокировки для СЧЕТ 3) 30 → 20
–	t5	Извлечение кортежа СЧЕТ 1: (задание S-блокировки для СЧЕТ 1)
–	t6	Обновление кортежа СЧЕТ 1: (задание X-блокировки для СЧЕТ 1) 40 → 50
–	t7	Ожидание
Извлечение кортежа СЧЕТ 3: (задание S-блокировки для СЧЕТ 3)	t8	Ожидание
Ожидание		Ожидание

рис. 0.10. Проблема несовместимого анализа разрешается, но в момент времени t7 возникает тупиковая ситуация.

10.6 Тупиковые ситуации

Как было показано выше, блокировку можно использовать для разрешения трех основных проблем, возникающих при параллельной обработке кортежей. К сожалению, использование блокировок приводит к возникновению другой проблемы – тупиковой ситуации. На

рис. 0.**11** показан обобщенный пример этой проблемы, в котором p1 и p2 представляют любые блокируемые объекты, необязательно кортежи базы данных, а выражения типа "блокировка ... без взаимного доступа" представляют любые операции с наложением блокировки без взаимного доступа, заданные как явно, так и неявно.

Транзакция А	Время	Транзакция В
Блокировка p1 без взаимного доступа	t1	–
–	t2	Блокировка p2 без взаимного доступа
Блокировка p2 без взаимного доступа	t3	–
Ожидание	t4	Блокировка p1 без взаимного доступа
Ожидание		Ожидание

рис. 0.11. Пример тупиковой ситуации.

Тупиковая ситуация возникает тогда, когда две или более транзакции одновременно находятся в состоянии ожидания, причем для продолжения работы каждая из транзакций ожидает прекращения выполнения другой транзакции.

Для обнаружения тупиковой ситуации следует обнаружить цикл в диаграмме состояний ожидания, т.е. в перечне "транзакций, которые ожидают окончания выполнения других транзакций" Поиск выхода из тупиковой ситуации состоит в выборе одной из заблокированных транзакций в качестве жертвы и отмене ее выполнения. Таким образом, с нее снимается блокировка, а выполнение другой транзакции может быть возобновлено.

На практике не все системы в состоянии обнаружить тупиковую ситуацию. Например, в некоторых из них используется хронометраж выполнения транзакций, и сообщение о возникновении тупиковой ситуации поступает, если транзакция не выполняется за некоторое предписанное заранее время.

Следует обратить внимание на то, что транзакция-жертва признается "некорректной" и отменяется "не из-за собственной некорректности". В некоторых системах предусмотрен автоматический перезапуск транзакции с самого начала при условии, что обстоятельства, которые привели к тупиковой ситуации, не повторятся вновь. А в других системах в программу, связанную с данной транзакцией, просто посылается сообщение о "вызвавшей тупиковую ситуацию транзакции-жертве" для обработки этой ситуации в самой программе С точки зрения программирования приложений предпочтительнее первый из этих подходов. Но несмотря на это, всегда рекомендуется решать данную проблему с точки зрения пользователя.

10.7 Способность к упорядочению

Чередующееся выполнение заданного множества транзакций будет верным, если оно упорядочено, т.е. при его выполнении будет получен такой же результат, как и при последовательное выполнении тех же транзакций. Обосновать это утверждение помогут следующие замечания:

1. Отдельные транзакции считаются верными, если при их выполнении база данных переходит из одного непротиворечивого состояния в другое непротиворечивое состояние.
2. Выполнение транзакций одна за другой в любом последовательном порядке также является верным. При этом под выражением "любой последовательный порядок" подразумевается, что используются независимые друг от друга транзакции.
3. Чередующееся выполнение транзакций, следовательно, является верным, если оно эквивалентно некоторому последовательному выполнению, т.е. если оно подлежит упорядочению.

Возвращаясь к приведенным выше примерам (

рис. 0.**2** –

рис. 0.**5**), можно отметить, что проблема в каждом случае заключалась в том, что чередующееся выполнение транзакций не было упорядочено, т.е. не было эквивалентно выполнению либо сначала транзакции А, а затем транзакции В, либо сначала транзакции В, а затем транзакции А.

Для заданного набора транзакций любой порядок их выполнения (чередующийся или какой-либо другой) называется графиком запуска. Выполнение транзакций по одной без их чередования называется последовательным графиком запуска, а непоследовательное выполнение транзакций – чередующимся графиком запуска или непоследовательным графиком запуска. Два графика называются эквивалентными, если при их выполнении будет получен одинаковый результат, независимо от исходного состояния базы данных. Таким образом, график запуска является верным (т.е. допускающим возможность упорядочения), если он эквивалентен некоторому последовательному графику запуска.

При выполнении двух различных последовательных графиков запуска, содержащих одинаковый набор транзакций, можно получить совершенно различные результаты. Поэтому выполнение двух различных чередующихся графиков запуска с одинаковыми транзакциями может также привести к различным результатам, которые могут быть восприняты как верные.

Теорема двухфазной блокировки (не имеет отношения к протоколу двухфазной фиксации), которая может быть сформулирована следующим образом:

Если все транзакции подчиняются "протоколу двухфазной блокировки", то для всех возможных чередующихся графиков запуска существует возможность упорядочения.

При этом протокол двухфазной блокировки, в свою очередь, формулируется следующим образом.

1. Перед выполнением каких-либо операций с некоторым объектом (например, с кортежем базы данных) транзакция должна заблокировать этот кортеж.
2. После снятия блокировки транзакция не должна накладывать никаких других блокировок.

Таким образом, транзакция, которая подчиняется этому протоколу, характеризуется двумя фазами: фазой наложения блокировки и фазой снятия блокировки.

Характеристика упорядочения может быть выражена следующим образом. Если A и B являются любыми двумя транзакциями некоторого графика запуска, допускающего возможность упорядочения, то либо A логически предшествует B, либо B логически предшествует A, т.е. либо B использует результаты выполнения транзакции A, либо A использует результаты выполнения транзакции B. (Если транзакция A приводит к обновлению кортежей p, q, ... r и транзакция B использует эти кортежи в качестве входных данных, то используются либо все обновленные с помощью A кортежи, либо полностью не обновленные кортежи до выполнения транзакции A, но никак не их смесь.) Наоборот, график запуска является неверным и не подлежит упорядочению, если результат выполнения транзакций не соответствует либо сначала выполнению транзакции A, а затем транзакции B, либо сначала выполнению транзакции B, а затем транзакции A.

В настоящее время с целью понижения требований к ресурсам и, следовательно, повышения производительности и пропускной способности в реальных системах обычно предусмотрено использование не двухфазных транзакций, а транзакций с "ранним снятием блокировки" (еще до выполнения операции прекращения транзакции) и наложением нескольких блокировок. Однако следует понимать, что использование таких транзакций сопряжено с большим риском. Действительно, при использовании недвухфазной транзакции A предполагается, что в данной системе не существует никакой другой чередующейся с ней транзакции B (в противном случае в системе возможно получение ошибочных результатов).

10.8 Уровни изоляции транзакции

Термин уровень изоляции, грубо говоря, используется для описания степени вмешательства параллельных транзакций в работу некоторой заданной транзакции. Но при обеспечении возможности упорядочения не допускается никакого вмешательства, иначе говоря, уровень изоляции должен быть максимальным. Однако, как уже отмечалось, в реальных системах по различным причинам обычно допускаются транзакции, которые работают на уровне изоляции ниже максимального.

Уровень изоляции обычно рассматривается как некоторое свойство транзакции. В реальных СУБД может быть реализовано различное количество уровней изоляции.

Кроме того помимо кортежей могут блокироваться другие единицы данных, например целое отношение, база данных или (пример противоположного характера) некоторое значение атрибута внутри заданного кортежа.

10.8.1 Поддержка в языке SQL

SQL поддерживает операции COMMIT и ROLLBACK для фиксации и отката транзакции соответственно.

Специальный оператор SET TRANSACTION используется для определения некоторых характеристик транзакции, которую нужно будет инициировать, такие, как режим доступа и уровень изоляции.

В стандарте языка SQL не предусмотрена поддержка явным образом возможности блокировки (фактически, блокировка в нем вообще не упоминается). Блокировки накладываются неявно, при выполнении операторов SQL.